ECSE 551 Mini-Project 1: Text Classification

Shankhin Brahmavar
260921778
shankhin.brahmavar@mail.mcgill.ca

Abstract

In this project we investigated the performance of different classification algorithms on posts posted in 5 different subreddits - from the public domain www.reddit.com. Namely, the 5 subreddits were - Instagram, Facebook, Viber, LinkedIn, Telegram and the algorithm were a Naive Bayes algorithm written from scratch and a Random Forest Classification algorithm from the SciKit Learn library applied on the dataset. There were several preprocessing steps taken in order to clean the data and have the data be as clean as possible from undesirable noise data so as to not hinder our prediction process and make it as accurate and precise as possible. We observed that the Naive Bayes classifier performed much worse (test accuracy of 60-70 percent) than the Random Forest Classifier (test accuracy of 85-95 percent). Additionally, we observed that fine tuning parameters like max features, stop words, strip accents in the Count Vectorizer class greatly affected test accuracies, especially in the Naive Bayes classifier.

1 Introduction

The task of this project was to classify the given texts in the csv file text.csv as accurately as possible after training the given models on the train.csv datasets. The train.csv dataset contained 2 columns - first one being the post from the subreddit mentioned in the corresponding 2nd column. After this we were meant to predict which subreddit the post in the test.csv file belonged to and finally, write the results onto a csv file called Submission.csv. We started this off by preprocessing the data by applying stopwords, stemming, stripping accents and making sure the data only contained letters - no numbers or symbols. After which we vectorized the data and assigned a number to each class (which were the subreddits). Then we created a Naive Bayes classifier class and used the Random Forest classifier from sklearn to predict and fit the data. Then we applied 10-fold cross validation to determine which had the better validation accuracy so as to decide which classifier to decide which was the better classifier. Finally, the predictions were written into a csv file.

2 Dataset

On of the main observations made on the dataset was that train.csv had evenly distributed data as we observed that the prior probability, P(Y), for all the 5 classes were very close to 0.2 hence there was a high probability that the posterior probability end up being an edge case - where it has equal probability of being in any of the classes hence the implementation of Laplace smoothing was high necessary in this case so as to ensure we get a definite answer as to which class the data would belong to. The data after preprocessing was also visualized through a wordcloud to give us a better idea of the text we're dealing with. This wordcloud can be seen in Figure 1. The bigger the word is, the higher it appears in the dataset. There were several preprocessing steps taken to ensure that the vectorized data would represent the meaning of the text as accurately as possible. First off, we applied stopwords from the sklearn library, then we stemmed the data using the PorterStemmer class from nltk library. Finally, we vectorized the data while stripping accents, applying variable max features

34th Conference on Neural Information Processing Systems (NeurIPS 2020), Vancouver, Canada.

Table 1: Accuracy for each model on a sample run

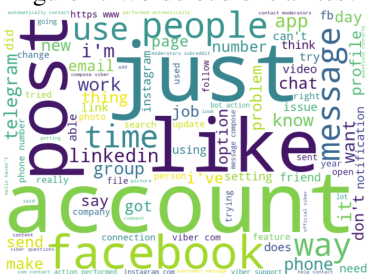
	Part	
Algorithm	Accuracy(Training,Test)	Ranking
RFC	100,89.33	1
NVB	92.33,66.07	2

and applying a token pattern rejecting characters that weren't letters. Finally, we mapped the class names to corresponding class labels that were numbers to enable easier processing of data during the actual training stage.

3 Proposed Approach

The approach we took in defining the Naive Bayes function was that of the one discussed in our lectures. The class was initialized with the prior probabilty P(Y) being calculated for the dataset as well as P(X|Y) being calculated for the dataset. After which the function predict in the class predicts the label for datapoints in the validation and test set using the probabilities of that certain datapoint being in class 0-5 and the max of these probabilities would be the label it would be predicted to have. The approach for the Random Forest Classifer was simple as it is predefined in the sklearn library. This classifier works by implementing several different decison trees and essentially averaging out the predicted class labels across these several decision trees. We observed that 600 decision trees is a sweet spot for operation time and algorithm accuracy as well as min samples split of 2.

Figure 1: Wordcloud of train.csv



4 Results

When running the algorithms on the dataset, we observed that in general, the Random Forest Classifier almost always outperformed the Naive Bayes classifier in terms of test accuracy due to its nature of overfitting to the data. Although, the Naive Bayes Classifier could contest when it came to training accuracies. We also observed that tampering with data by changing max features had a drastic effect on the performance of the Naive Bayes Classifier and that a good spot for it was in the 100 range as this allowed for higher variance and lower bias. Although it seemed like these changes did not affect the other classifier as much due to it being a higher variance model in general. A test run of their accuracies can be seen in Table 1.

5 Discussion and Conclusion

We concluded that the Random Forest Classifier is better than a normal Naive Bayes classifier in terms of this particular problem of text classification. Although it is important to note that perhaps the use of laplace smoothing would have made a better point for Naive Bayes as a classifier.

6 Statement of Contributions

Since I was the only one in my group, I did all of the work and research myself.

7 Appendix

```
from google.colab import drive
from wordcloud import WordCloud, STOPWORDS
from matplotlib import pyplot as plt
import csv
import random
import collections
import numpy as np
from sklearn.metrics import accuracy_score
from sklearn import svm
from sklearn import ensemble
import os
import re
import nltk
nltk.download('punkt')
nltk.download('wordnet')
nltk.download('averaged_perceptron_tagger')
drive.mount('/content/gdrive')
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     /root/nltk_data...
[nltk_data]   Package averaged_perceptron_tagger is already up-to-
[nltk_data]       date!
Drive already mounted at /content/gdrive; to attempt to forcibly remount, call drive.mou
```

Import Data

```
with open('/content/gdrive/MyDrive/Colab Notebooks/train.csv', encoding='mac_roman') as csv_f
  csv_reader = csv.reader(csv_file)
  colnames = next(csv_reader)  # skip column names
  X_train = []
  Y_train = []
  for row in csv_reader:
      X_train.append(row[0])
      Y_train.append(row[1])

with open('/content/gdrive/MyDrive/Colab Notebooks/test.csv', encoding='mac_roman') as csv_fi
  csv_reader = csv.reader(csv_file)
  colnames = next(csv_reader)  # skip column names
  X_test = []
  test_ID = []
  for row in csv_reader:
      X_test.append(row[1])
      test_ID.append(row[0])
```

Define stemming and stopwords function

```
from sklearn.feature_extraction import text
from nltk.stem import PorterStemmer

my_stop_words = text.ENGLISH_STOP_WORDS

def stem(post):
  ps = PorterStemmer()
  return " ".join([ps.stem(word) for word in post.split(' ')])

def stopwords(post):
  return " ".join([word for word in post.split(' ') if not word in my_stop_words])


X_train = [stem(stopwords(x)) for x in X_train]
X_test = [stem(stopwords(x)) for x in X_test]
```

Vectorize training and test data

```
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction import text
my_stop_words = text.ENGLISH_STOP_WORDS
vectorizer = CountVectorizer(max_features=1000, strip_accents='ascii', binary = 'true', token
X_train_vect = vectorizer.fit_transform(X_train)
X_test_vect = vectorizer.transform(X_test)
print(vectorizer.get_feature_names())
```

```
['abil', 'abl', 'about', 'above', 'absolut', 'accept', 'access', 'accident', 'account',
```

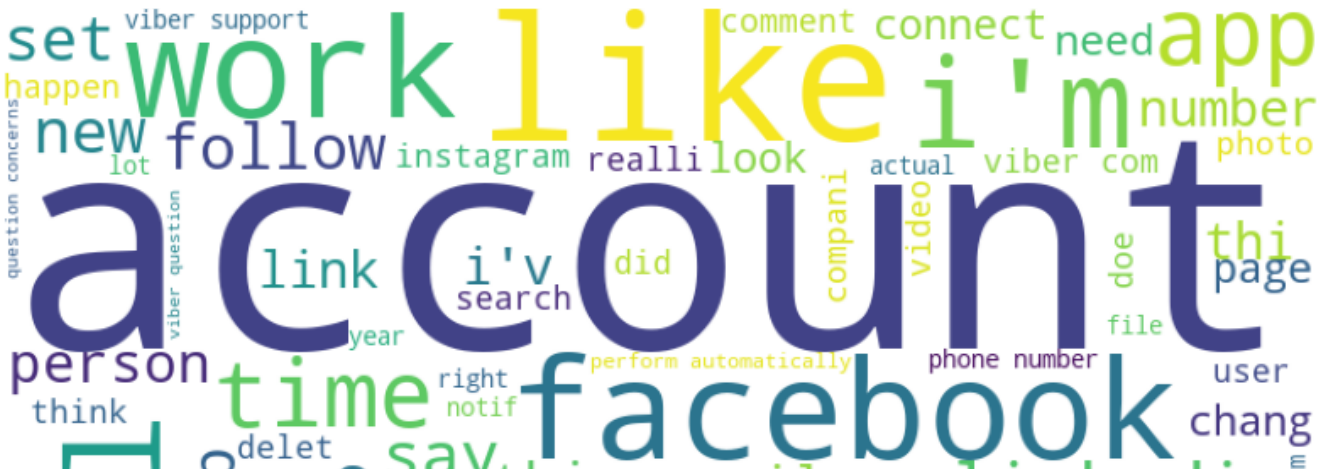
Enumeration to change y data to integers

```
from enum import Enum

class Classes(Enum):
    Telegram = 0
    Linkedin = 1
    Instagram = 2
    viber = 3
    Facebook = 4

Y_train_vect = [getattr(Classes, n).value for n in Y_train]
```

Create wordcloud to visualize data

```
#wordcloud of data
text = ' '
for x in X_train:

    x = str(x)

    values = x.split()

    for i in range(len(values)):
        values[i] = values[i].lower()

    for words in values:
        text = text + words + ' '


wc = WordCloud(max_words= 100,
                      width = 744,
                      height = 544,
                      background_color ='white',
                      stopwords= my_stop_words,
                      contour_width=3,
                      contour_color='steelblue',
                      min_font_size = 10).generate(text)

plt.figure(figsize = (14, 14))
plt.imshow(wc)
plt.axis("off")
```

```
(-0.5, 743.5, 543.5, -0.5)
```



Naive-Bayes class

```
class NVB:

  def __init__ (self, X_train_vect, Y_train_vect):
    counts = collections.Counter(Y_train_vect)
    self.prior = [counts[index]/len(Y_train_vect) for index in counts]

    self.posterior = []
    for i in range(5):
      sum = np.squeeze(np.asarray((np.divide(np.sum(np.array([X_train_vect[index] for index i
      self.posterior.append(np.column_stack((np.subtract(1,sum),sum)))

  def predict(self, X_test_vect):
    predictions = []
    for post in X_test_vect:
      problist = [self.probability(post,y) for y in range(5)]
      predictions.append(max(range(5), key=problist.__getitem__))
    return predictions

  def probability(self,post, y):
    pi = self.prior[y]
    for index, feature in enumerate(post):
      pi *= self.posterior[y][index][feature]
    return pi


model = NVB(X_train_vect, Y_train_vect)
model_predictions = model.predict(np.squeeze(np.asarray(X_train_vect.todense())))
print("Train accuracy:", accuracy_score(Y_train_vect,model_predictions))
```

```
Train accuracy: 0.9228130360205832
```

Random Forest Classifier from sklearn

```
rfc = ensemble.RandomForestClassifier(n_estimators = 1000, min_samples_split = 2, max_depth =
rfc.fit(X_train_vect, Y_train_vect)

train_predict_rfc = rfc.predict(X_train_vect)
print("Train accuracy:", accuracy_score(Y_train_vect,train_predict_rfc))
```

```
Train accuracy: 1.0
```

10-fold cross validation for Naive Bayes Classifier

```
from sklearn.model_selection import KFold
kf = KFold(n_splits=10, shuffle = True)
kf.get_n_splits(X_train_vect)

accuracies = []
counter = 0
for train_index, test_index in kf.split(X_train_vect):
  newX_train, newX_test = X_train_vect[train_index], X_train_vect[test_index]
  newY_train, newY_test = np.expand_dims(Y_train_vect, axis = 1)[train_index], np.expand_dims
  newY_train = np.squeeze(np.asarray(newY_train))
  model = NVB(newX_train, newY_train)
  model_predictions = model.predict(np.squeeze(np.asarray(newX_test.todense())))
  print("Test accuracy:", counter , accuracy_score(newY_test,model_predictions))
  accuracies.append(accuracy_score(newY_test,model_predictions))
  counter += 1
print("Cross Validate Accuracy:", np.mean(accuracies))
```

```
Test accuracy: 0 0.6285714285714286
Test accuracy: 1 0.6228571428571429
Test accuracy: 2 0.6742857142857143
Test accuracy: 3 0.7028571428571428
Test accuracy: 4 0.6914285714285714
Test accuracy: 5 0.6057142857142858
Test accuracy: 6 0.7028571428571428
Test accuracy: 7 0.68
Test accuracy: 8 0.6742857142857143
Test accuracy: 9 0.6781609195402298
Cross Validate Accuracy: 0.6661018062397372
```

10-fold cross validation for Random Forest Classifier

```
from sklearn.model_selection import KFold
kf = KFold(n_splits=10, shuffle = True)
kf.get_n_splits(X_train_vect)

accuracies = []
counter = 0
```

```
for train_index, test_index in kf.split(X_train_vect):
  newX_train, newX_test = X_train_vect[train_index], X_train_vect[test_index]
  newY_train, newY_test = np.expand_dims(Y_train_vect, axis = 1)[train_index], np.expand_dims
  newY_train = np.squeeze(np.asarray(newY_train))
  rfc = ensemble.RandomForestClassifier(n_estimators = 600, min_samples_split = 2, max_depth
  rfc.fit(newX_train, newY_train)
  train_predict_rfc = rfc.predict(newX_test)
  print("Test accuracy:", counter , accuracy_score(newY_test,train_predict_rfc))
  accuracies.append(accuracy_score(newY_test,train_predict_rfc))
  counter += 1
print("Cross Validate Accuracy:", np.mean(accuracies))
```

```
Test accuracy: 0 0.8742857142857143
Test accuracy: 1 0.8571428571428571
Test accuracy: 2 0.8914285714285715
Test accuracy: 3 0.9142857142857143
Test accuracy: 4 0.8914285714285715
Test accuracy: 5 0.88
Test accuracy: 6 0.8742857142857143
Test accuracy: 7 0.8742857142857143
Test accuracy: 8 0.88
Test accuracy: 9 0.867816091954023
Cross Validate Accuracy: 0.880495894909688
```

```
rfc.fit(X_train_vect,Y_train_vect)
test_predict_vect = rfc.predict(X_test_vect)
test_predict = [Classes(num).name for num in test_predict_vect]
```

```
['Telegram', 'Linkedin', 'Telegram', 'Telegram', 'Telegram', 'Telegram', 'Linkedin', 'Te
```

Writing into csv file

```
counter = 1
tmpFile = '/content/gdrive/MyDrive/Colab Notebooks/Submission.csv'
with open(tmpFile, "w") as file:
    writer = csv.writer(file, delimiter=',')
    for i in test_predict:
      row = [counter,i]
      writer.writerow(row)
      counter+=1
```

✓ 0s completed at 10:15 PM